Lambda Expressions in Java 8 Sessions Report

Arkady Galyash

dxChartPro

October 24, 2012

Agenda

1. Lambda Intro
2. Lambda Expression
3. Interface Evolution
4. Library Changes
5. Lambda in JVM
6. Links

Why Lambda Project is so popular?

- Huge language and library improvements
- Perhaps the biggest upgrade ever to the Java programming model
- We want to treat code as data

What is a Lambda Expression?

Lambda expression(closure) is an anonymous method

- has an argument list, a return type, a body
- can refer values from the enclosing lexical scope
- not a member of a class - just free-floating expression

Who needs Lambda Expressions?

- C++ added them
- C# added them
- Scala, Groovy, Clojure, Kotlin, ...

> In another thirty years people will laugh at anyone who tries to invent a language without closures, just as they'll laugh now at anyone who tries to invent a language without recursion.
>
> — Mark Jason Dominus, "The Perl Review"

How Lambda Expression looks like?

Talk is cheap. Show me the code.

What have been done?

JSR-335 =

- Lambda Expression
- Interface Evolution
- Bulk Collection Operations

Study calculation example

- Chart with some studies
- Study can be calculated
- Some studies are visible and some are hidden
- New bars have come - we need to recalculate studies

What is changed?

- Library is in control(parallelism, laziness)
- Behavior have been passed into the API as data
- Internal iteration
- More what, less how

How much syntax sugar do we have?

- Remove curly brackets and return for single expression
- Type argument inference
- Method reference
  `Classname::methodName`
- Expression method reference
  `Expression::methodName`
- Effectively final local variables

What about type?

- Now we are using single-method interfaces to represent functions:
  Runnable, Comparator, ActionListener
- Let's give these a name: functional interfaces
- We need more functional interfaces:
  Predicate, Block, etc.
- Everything we wrote before will work with lambda for free

Interface evolution

`Collection::forEach`

`(Block<? super E>)`

What is this?

Interface inheritance rules

- ArrayList VS List

Interface inheritance rules

- ArrayList > List "superclass always wins"

Interface inheritance rules

- ArrayList > List "superclass always wins"
- List VS Collection

Interface inheritance rules

- ArrayList > List "superclass always wins"
- List > Collection "subtype wins"

Interface inheritance rules

- ArrayList > List "superclass always wins"
- List > Collection "subtype wins"
- List VS Set

Interface inheritance rules

- ArrayList > List "superclass always wins"
- List > Collection "subtype wins"
- List == Set write your implementation

Diamond problem

java.util.Collection

- `void forEach(Block<? super E>)`
- `boolean removeIf(Predicate<? super E>)`
- `Stream<E> stream()`
- `Stream<E> parallel()`

java.util.stream.Stream

Methods for FP-lovers

- `<R> Stream<R> map(Mapper<? super T, ? extends R> mapper);`
- `<R> Stream<R> flatMap(FlatMapper<? super T, R> mapper);`
- `Stream<T> filter(Predicate<? super T> predicate);`

java.util.stream.Stream #2

Terminal methods

- `<U> U fold(Factory<U> baseFactory, Combiner<U, T, U> reducer, BinaryOperator<U> combiner);`
- `T reduce(T base, BinaryOperator<T> op);`
- `Optional<T> reduce(BinaryOperator<T> op);`
- `<A extends Destination<? super T>> A into(A target);`

Parallel Stream

- Stream<E> parallel()
- Over fork/join framework
- java.util.streams.Spliterator provides the ability to decompose an aggregate data structure and to iterate over the elements of the aggregate

Lambda in JVM

Lambda is just sugar?

We could say that a lambda is "just" an inner class instance.

Pros:

- Simple, quick and dirty
- We use that already worked

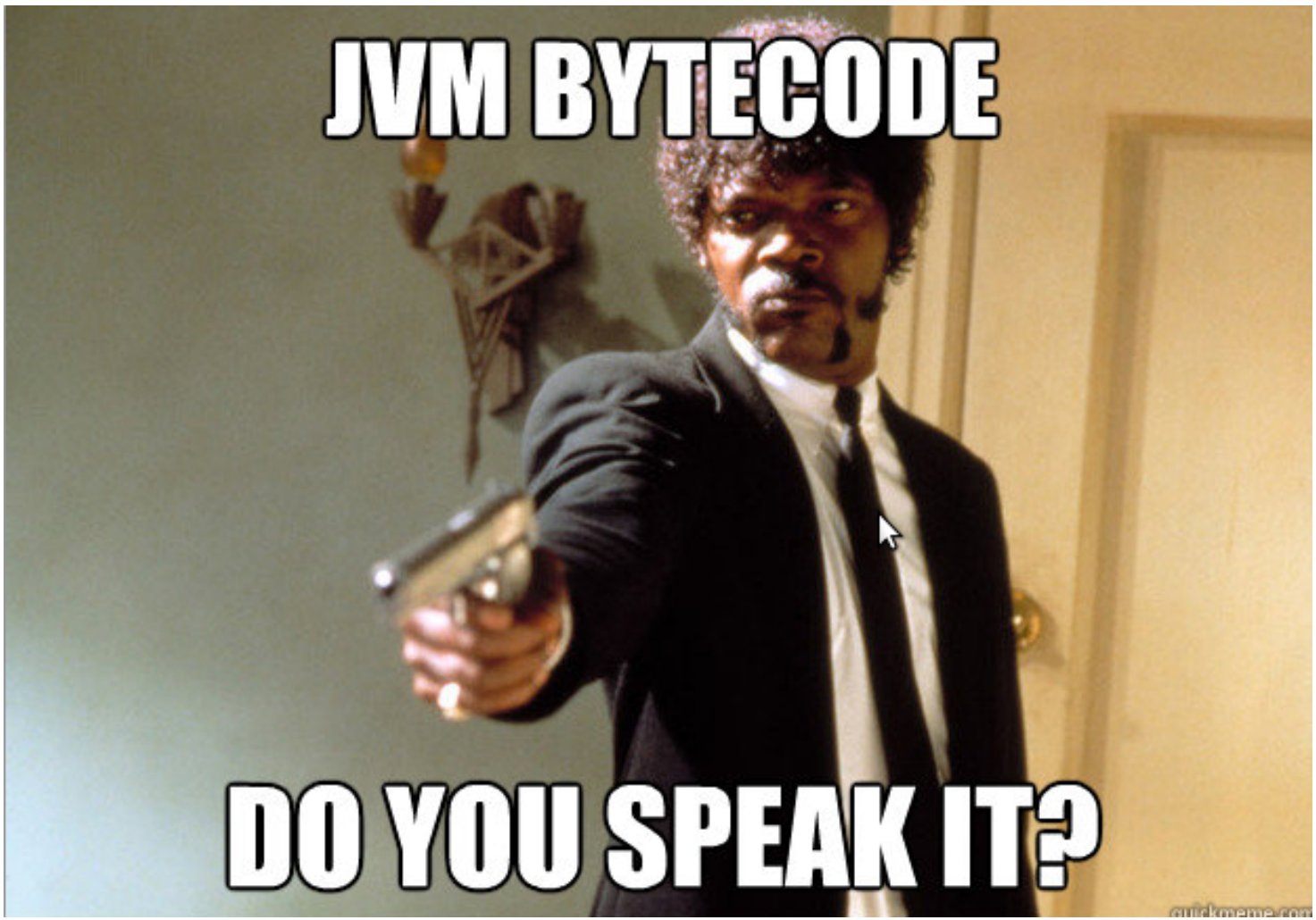
Lambda is just sugar?

We could say that a lambda is "just" an inner class instance.

Cons:

- One class per lambda expression(performance issues)
- Whatever we do becomes a binary representation for lambdas in Java(forever)

JVM Bytecode



Bytecode invocation modes

Prior to Java 7, the JVM had four bytecodes for method invocation

- invokestatic - for static methods
- invokevirtual - for class methods
- invokeinterface - for interface methods
- invokespecial - for everything else

java.lang.invoke.MethodHandle

- From Java7
- Typed, directly executable reference to an underlying method or similar low-level operation, with optional transformations of arguments or return values
- Can be saved into runtime constant pool of the class and loaded with LDC
- Sounds like lambda

java.lang.invoke.MethodHandle

```
list.removeAll(...)
```

```
private static boolean lambda$1(...) { ... }
```

```
MethodHandle mh = LDC[lambda$1];
mh = mh.insertArguments(mh, ...);
list.removeAll(mh);
```

java.lang.invoke.MethodHandle

Cons:

- Signature will be
  void removeAll(MethodHandle predicate)
- Erasure worse than with generics(all lambda are the same)
- Whatever we do becomes a binary representation for lambdas in Java(forever)

invokedynamic

- From Java7
- Let some "language logic" determinate call target
- Language and VM become partners in flexible and efficient method dispatch
- java.lang.invoke.CallSite

indy and Lambda

- indy lets us separate the binary representation of lambda creation in the bytecode from the mechanics of evaluating the lambda expression at runtime
- Instead of generating bytecode to create the object that implements the lambda expression (such as calling a constructor for an inner class), we describe a recipe for constructing the lambda, and delegate the actual construction to the language runtime

java.lang.invoke.LambdaMetafactory

- Bootstrap for the lambda factory selects the translation strategy
- The runtime implementation choice is hidden behind a standardized API for lambda construction(invokedynamic)

java.lang.invoke.LambdaMetafactory

Talk is cheap. Show me the ~~code~~ bytecode.

Desugaring Lambda

- For "stateless" lambda - generate private static method at the same class with the signature of SAM
- For lambda capturing immutable values - generate private static method at the same class with the signature of SAM + additional captured parameters
- For lambda with this, super, etc. - generate private method of the same class with the signature of SAM

Lambda Serialization

Lambda Serialization

- Dynamic translation strategy mandates a dynamic serialization strategy
- The serialized form(java.lang.invoke.SerializedLambda) would have to contain all the information needed to recreate the object through the metafactory

HowTo play with Lambda?

- Project Lambda
- hg repository
- Binary snapshots
- -XDlambdaToMethod

What to read about Lambda?

- State of the Lambda v4
- Defender methods v4
- State of the Lambda: Libraries Edition
- Translation of Lambda Expressions
- CON4862, CON6080

Thank You!